```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
%matplotlib inline
from  google.colab import files
import io
```

```
data=files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Admission_Predict.csv to Admission_Predict.csv
```

```
data=pd.read_csv('/content/Admission_Predict.csv')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

```
data.isnull().any()
```

```
Serial No.           False
GRE Score            False
TOEFL Score          False
University Rating    False
SOP                  False
LOR                  False
CGPA                 False
Research             False
Chance of Admit      False
dtype: bool
```

```
data=data.rename(columns ={'Chance of Admit ': 'Chance of Admit','LOR ':'LOR'})
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
```

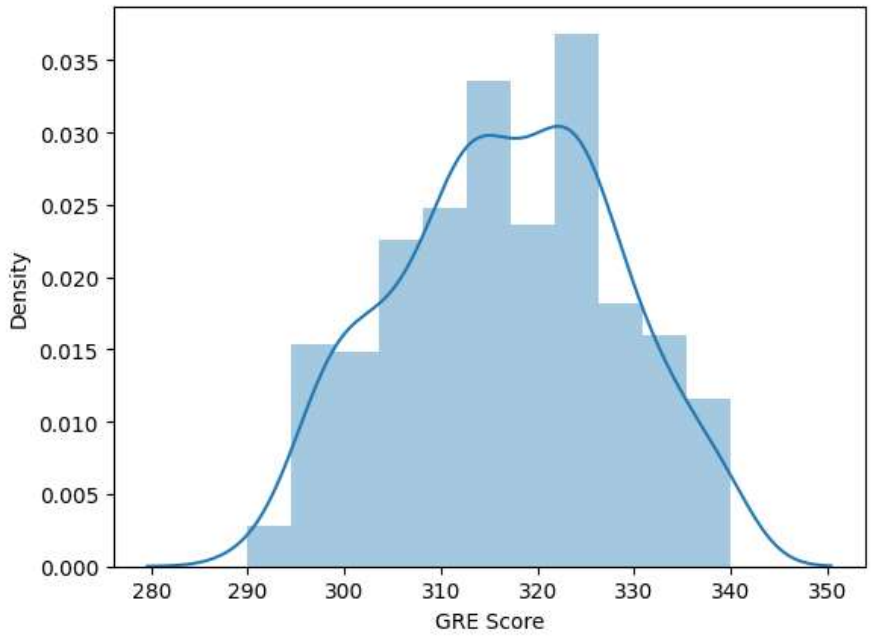
```
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

```
data.describe()
```

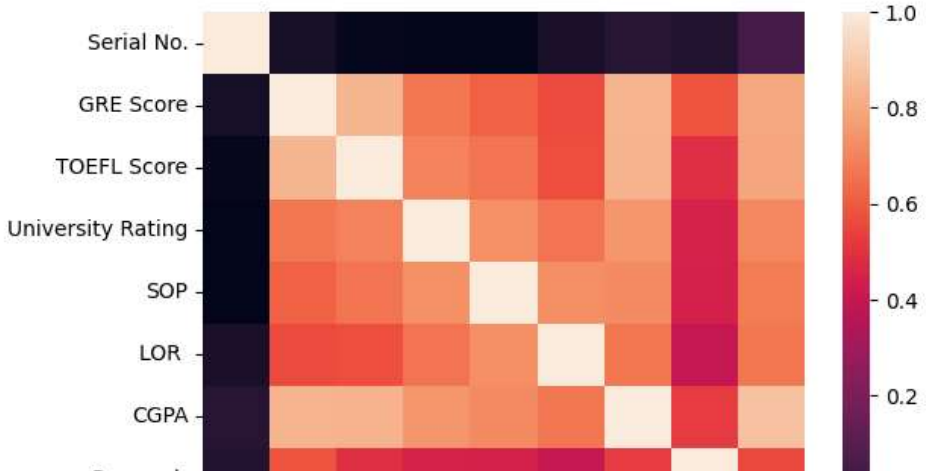
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CG
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.0000
mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.5989
std	115.614301	11.473646	6.069514	1.143728	1.006869	0.898478	0.5963
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.000000	6.8000
25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.1700
50%	200.500000	317.000000	107.000000	3.000000	3.500000	3.500000	8.6100
75%	300.250000	325.000000	112.000000	4.000000	4.000000	4.000000	9.0625
max	400.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.9200

```
sns.distplot(data["GRE Score"])
```

```
<ipython-input-12-35a15aaea6d8>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data["GRE Score"])
<Axes: xlabel='GRE Score', ylabel='Density'>
```

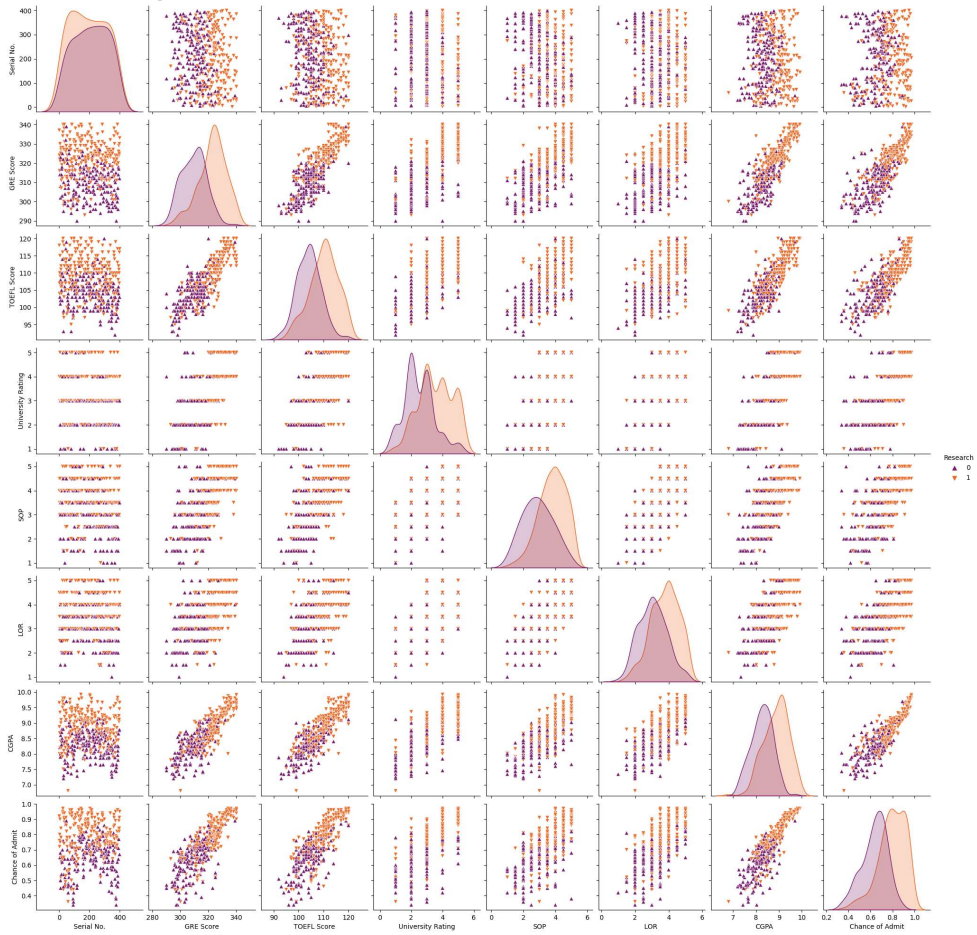



```
ax = sns.heatmap(data.corr(), annot=False)
```



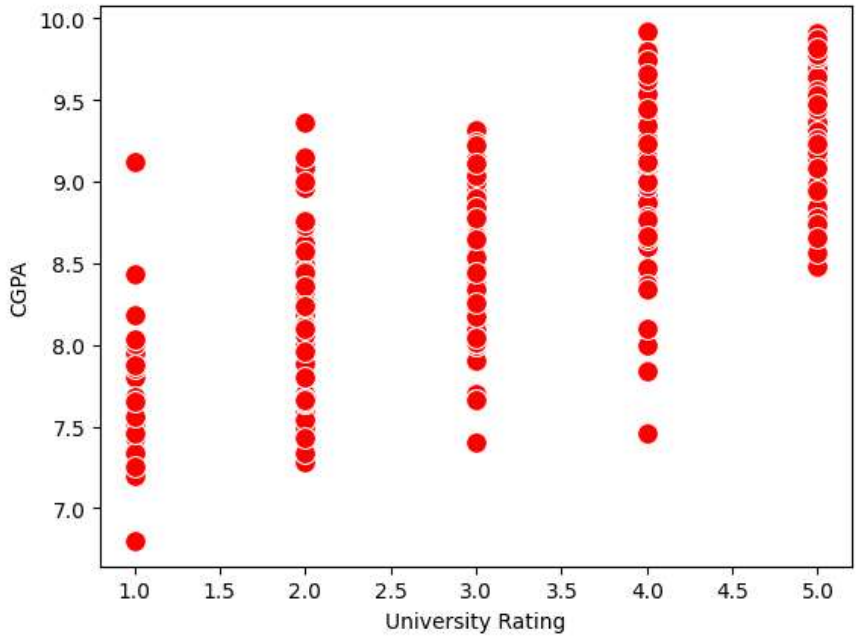

```
sns.pairplot(data=data, hue="Research", markers=["^", "v"],palette="inferno")
```

```
<seaborn.axisgrid.PairGrid at 0x7f4f5dc6fa30>
```



```
sns.scatterplot(x="University Rating",y="CGPA",data=data,color="Red",s=100)
```

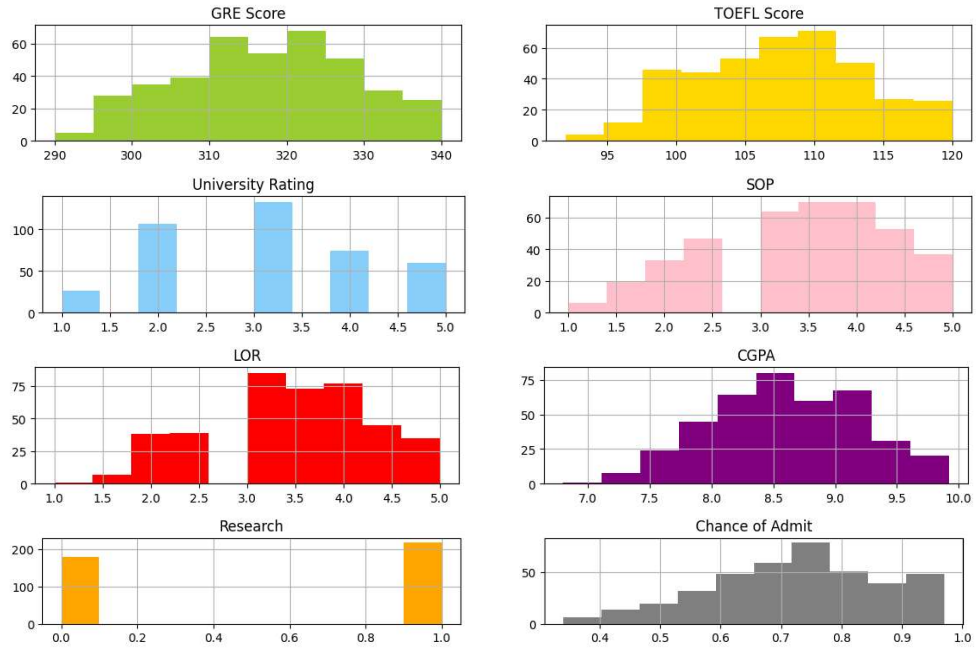
```
<Axes: xlabel='University Rating', ylabel='CGPA'>
```




```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

```
category=["GRE Score",'TOEFL Score','University Rating','SOP','LOR','CGPA','Research','Chance of Admit']
color=["yellowgreen","gold","lightskyblue",'pink','red','purple','orange','grey']
start=True
for i in np.arange(4):
  fig=plt.figure(figsize=(14,8))
  plt.subplot2grid((4,2),(i,0))
  data[category[2*i]].hist(color=color[2*i],bins=10)
  plt.title(category[2*i])
  plt.subplot2grid((4,2),(i,1))
  data[category[2*i+1]].hist(color=color[2*i+1],bins=10)
  plt.title(category[2*i+1])

plt.subplots_adjust(hspace=0.7,wspace=0.2)
plt.show()
```




```
x=data.iloc[:,0:7].values
x
```

```
array([[  1.  , 337.  , 118.  , ...,   4.5 ,   4.5 ,   9.65],
       [  2.  , 324.  , 107.  , ...,   4.  ,   4.5 ,   8.87],
       [  3.  , 316.  , 104.  , ...,   3.  ,   3.5 ,   8.  ],
       ...,
       [398.  , 330.  , 116.  , ...,   5.  ,   4.5 ,   9.45],
       [399.  , 312.  , 103.  , ...,   3.5 ,   4.  ,   8.78],
       [400.  , 333.  , 117.  , ...,   5.  ,   4.  ,   9.66]])
```

```
y=data.iloc[:,7].values
y
```

```
array([1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1])
```

```
from sklearn.preprocessing import MinMaxScaler
sc=MinMaxScaler()
x_sc=sc.fit_transform(x)
x_sc
```

```
array([[0.        , 0.94      , 0.92857143, ..., 0.875     , 0.875     ,
        0.91346154],
       [0.00250627, 0.68      , 0.53571429, ..., 0.75      , 0.875     ,
        0.66346154],
       [0.00501253, 0.52      , 0.42857143, ..., 0.5       , 0.625     ,
        0.38461538],
       ...,
```

```
            [0.99498747, 0.8       , 0.85714286, ..., 1.        , 0.875     ,
             0.84935897],
            [0.99749373, 0.44      , 0.39285714, ..., 0.625     , 0.75      ,
             0.63461538],
            [1.        , 0.86      , 0.89285714, ..., 1.        , 0.75      ,
             0.91666667]])
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=101)
```

```
y_train=(y_train>0.5)
y_train
```

```
array([ True, False,  True,  True,  True,  True,  True,  True, False,
        True,  True, False,  True,  True, False, False,  True, False,
        True, False, False, False, False,  True,  True,  True,  True,
        True, False, False,  True, False, False, False, False,  True,
        True, False,  True, False, False,  True,  True, False,  True,
        True, False,  True,  True, False,  True,  True, False,  True,
       False,  True, False, False,  True, False,  True,  True,  True,
       False, False, False,  True, False, False,  True,  True, False,
        True, False,  True,  True, False,  True,  True,  True, False,
        True, False, False,  True,  True, False,  True,  True, False,
       False, False, False,  True,  True, False, False, False,  True,
       False, False, False, False,  True, False, False, False,  True,
       False, False,  True,  True,  True, False,  True,  True,  True,
        True,  True,  True,  True,  True, False, False,  True,  True,
        True,  True,  True,  True,  True,  True, False, False, False,
       False, False,  True,  True, False, False, False,  True,  True,
        True,  True, False,  True,  True, False, False, False,  True,
       False,  True,  True,  True,  True,  True, False,  True, False,
       False, False,  True,  True,  True,  True,  True, False,  True,
       False, False, False,  True,  True,  True,  True,  True,  True,
       False,  True, False, False,  True,  True,  True,  True,  True,
       False, False,  True,  True,  True,  True, False, False, False,
        True,  True,  True,  True,  True,  True, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False,  True, False,  True, False,
       False, False,  True, False, False,  True,  True,  True,  True,
        True,  True, False,  True, False,  True, False,  True,  True,
        True,  True,  True,  True, False, False,  True, False,  True,
        True,  True,  True,  True, False,  True, False, False,  True,
        True, False,  True,  True, False,  True, False,  True,  True,
        True])
```

```
y_test=(y_test>0.5)
y_test
```

```
array([False, False,  True, False,  True,  True, False, False,  True,
       False, False,  True,  True,  True, False, False,  True, False,
       False, False, False, False,  True,  True, False,  True, False,
        True, False,  True,  True,  True, False,  True, False, False,
       False, False, False, False,  True, False,  True,  True,  True,
        True, False, False, False,  True, False,  True,  True, False,
        True,  True, False,  True, False, False, False,  True, False,
       False, False, False, False,  True, False, False, False, False,
        True, False, False,  True,  True,  True, False, False,  True,
        True, False,  True,  True,  True, False,  True,  True, False,
       False, False,  True,  True, False, False,  True,  True, False,
       False, False, False, False, False, False,  True, False,  True,
        True,  True, False, False, False, False,  True, False, False,
        True,  True, False])
```

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(random_state=0)
l=lr.fit(x_train,y_train)
y_pred=lr.predict(x_test)
l
y_pred
```

```
array([False, False, False, False, False,  True,  True, False,  True,
        True,  True, False,  True,  True, False,  True,  True, False,
       False, False, False, False,  True, False, False,  True,  True,
        True, False,  True,  True,  True, False,  True, False, False,
```

```
              False, False,  True,  True, False,  True,  True,  True,  True,
               True, False,  True, False,  True, False,  True,  True, False,
               True,  True,  True, False, False, False, False,  True, False,
              False,  True, False, False, False, False, False, False,  True,
               True,  True,  True,  True, False,  True, False, False,  True,
               True, False,  True,  True,  True,  True, False,  True, False,
              False, False,  True,  True, False, False, False,  True, False,
               True, False, False,  True,  True, False,  True, False,  True,
               True,  True,  True,  True, False,  True,  True,  True, False,
              False,  True,  True])
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation,Dropout

classifier=keras.Sequential()
classifier.add(Dense(7,activation='relu',input_dim=7))

classifier.add(Dense(7,activation='relu'))
classifier.add(Dense(7,activation='linear'))

classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 56

 dense_1 (Dense)             (None, 7)                 56

 dense_2 (Dense)             (None, 7)                 56

=================================================================
Total params: 168
Trainable params: 168
Non-trainable params: 0
_________________________________________________________________
```

```
loss_1=tf.keras.losses.BinaryCrossentropy()
classifier.compile(loss=loss_1,optimizer="Adam",metrics=['accuracy'])
classifier.fit(x_train,y_train,batch_size=20,epochs=100)
```

```
Epoch 1/100
14/14 [==============================] - 1s 4ms/step - loss: 3.0168 - accuracy: 0.2893
Epoch 2/100
14/14 [==============================] - 0s 4ms/step - loss: 3.0140 - accuracy: 0.3071
Epoch 3/100
14/14 [==============================] - 0s 5ms/step - loss: 3.0116 - accuracy: 0.3179
Epoch 4/100
14/14 [==============================] - 0s 3ms/step - loss: 3.0222 - accuracy: 0.3214
Epoch 5/100
14/14 [==============================] - 0s 3ms/step - loss: 3.0202 - accuracy: 0.3321
Epoch 6/100
14/14 [==============================] - 0s 3ms/step - loss: 3.0193 - accuracy: 0.3357
Epoch 7/100
14/14 [==============================] - 0s 4ms/step - loss: 3.0193 - accuracy: 0.3393
Epoch 8/100
14/14 [==============================] - 0s 4ms/step - loss: 3.0177 - accuracy: 0.3393
Epoch 9/100
14/14 [==============================] - 0s 3ms/step - loss: 3.0176 - accuracy: 0.3357
Epoch 10/100
14/14 [==============================] - 0s 4ms/step - loss: 3.0164 - accuracy: 0.3429
Epoch 11/100
14/14 [==============================] - 0s 3ms/step - loss: 3.0158 - accuracy: 0.3500
Epoch 12/100
14/14 [==============================] - 0s 4ms/step - loss: 3.0153 - accuracy: 0.3571
Epoch 13/100
14/14 [==============================] - 0s 3ms/step - loss: 3.0147 - accuracy: 0.3571
Epoch 14/100
14/14 [==============================] - 0s 8ms/step - loss: 3.0131 - accuracy: 0.3536
Epoch 15/100
14/14 [==============================] - 0s 12ms/step - loss: 3.0133 - accuracy: 0.3536
Epoch 16/100
14/14 [==============================] - 0s 7ms/step - loss: 3.0130 - accuracy: 0.3536
Epoch 17/100
```

```
14/14 [==============================] - 0s 7ms/step - loss: 3.0121 - accuracy: 0.3571
Epoch 18/100
14/14 [==============================] - 0s 9ms/step - loss: 3.0113 - accuracy: 0.3571
Epoch 19/100
14/14 [==============================] - 0s 8ms/step - loss: 3.0111 - accuracy: 0.3571
Epoch 20/100
14/14 [==============================] - 0s 9ms/step - loss: 3.0101 - accuracy: 0.3571
Epoch 21/100
14/14 [==============================] - 0s 7ms/step - loss: 3.0113 - accuracy: 0.3571
Epoch 22/100
14/14 [==============================] - 0s 5ms/step - loss: 3.0145 - accuracy: 0.3607
Epoch 23/100
14/14 [==============================] - 0s 2ms/step - loss: 3.0164 - accuracy: 0.3536
Epoch 24/100
14/14 [==============================] - 0s 3ms/step - loss: 3.0106 - accuracy: 0.3536
Epoch 25/100
14/14 [==============================] - 0s 2ms/step - loss: 3.0097 - accuracy: 0.3536
Epoch 26/100
14/14 [==============================] - 0s 3ms/step - loss: 3.0080 - accuracy: 0.3571
Epoch 27/100
14/14 [==============================] - 0s 2ms/step - loss: 3.0076 - accuracy: 0.3571
Epoch 28/100
14/14 [==============================] - 0s 3ms/step - loss: 3.0066 - accuracy: 0.3571
Epoch 29/100
14/14 [==============================] - 0s 2ms/step - loss: 3.0063 - accuracy: 0.3571
```

```
from sklearn.metrics import accuracy_score
t_p=classifier.predict(x_train)
print(t_p)
```

```
9/9 [==============================] - 0s 2ms/step
[[ 1.0459894  -0.5715048   1.0673568  ...  1.0661731  -0.41084728
   1.0874926 ]
 [ 0.91784036 -0.6004993   0.9204847  ...  0.9492643  -0.2988746
   0.9606166 ]
 [ 0.62901646 -0.38047814  0.6068774  ...  0.6049504  -0.23528156
   0.6087853 ]
 ...
 [ 1.0695648  -0.6080221   1.0918226  ...  1.0995339  -0.40376195
   1.1204951 ]
 [ 0.9804415  -0.5233968   0.99607295 ...  0.98854876 -0.39520746
   1.0080665 ]
 [ 0.80103683 -0.50781476  0.7938847  ...  0.8089706  -0.27560815
   0.81747675]]
```

```
tr_acc=classifier.evaluate(x_train,y_train,verbose=0)[1]
print(tr_acc)
```

```
0.31785714626312256
```

```
test_acc=classifier.evaluate(x_test,y_test,verbose=0)[1]
print(test_acc)
```

```
0.4833333194255829
```

```
from sklearn.metrics import accuracy_score,recall_score, roc_auc_score, confusion_matrix,classification_report
print("Accuracy Score : %f" %(accuracy_score(y_test,y_pred)*100))
print("Recall Score : %f" %(recall_score(y_test,y_pred)*100))
print("ROC Score : %f" %(roc_auc_score(y_test,y_pred)*100))
print("\nConfusion Matrix")
print(confusion_matrix(y_test,y_pred))
print("\nClassification Report")
print(classification_report(y_test,y_pred))
```

```
Accuracy Score : 71.666667
Recall Score : 78.431373
ROC Score : 72.549020

Confusion Matrix
[[46 23]
 [11 40]]

Classification Report
              precision    recall  f1-score   support
```

```
              False       0.81      0.67      0.73        69
               True       0.63      0.78      0.70        51

           accuracy                           0.72       120
          macro avg       0.72      0.73      0.72       120
       weighted avg       0.73      0.72      0.72       120
```

```
classifier.save("IAdmission.h5")
```